Data Distribution & Processing CSCI
Data Fusion CSC
Requirements Design Panel 3

August 8 , 1997
Version 1.0

1. Data Distribution & Processing CSCI

The Data Distribution & Processing CSCI is composed of the following CSCs:

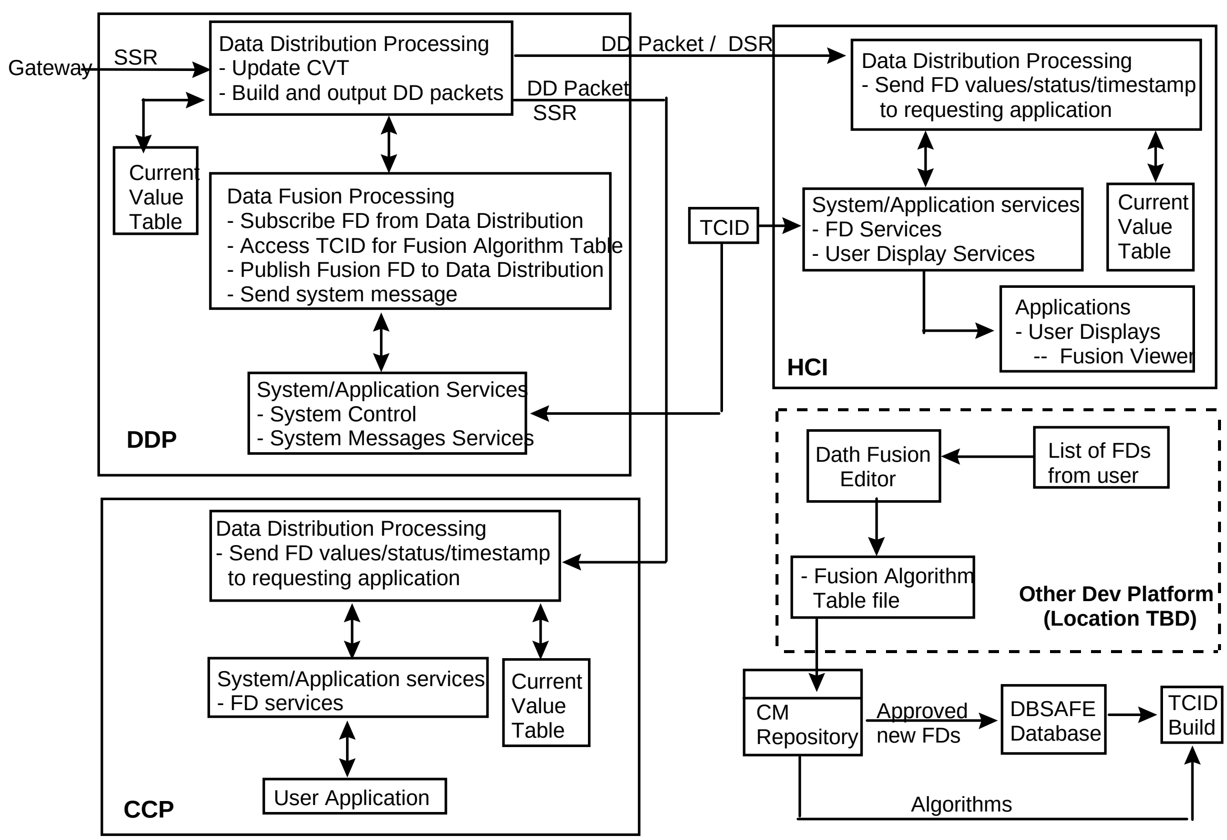
Data Distribution CSC, Data Fusion CSC, and Data Health CSC.

1.1 Data Fusion CSC Introduction

1.1.1 Data Fusion CSC Overview

Data Fusion involves computation using constants, FD measurement values, health values or other fusion values.   The Data Fusion CSC provides the capability to perform Data Fusion processing on specified FDs received at the DDP, based on algorithms defined for the associated FDs.

Data Fusion CSC Overview is as follows:



1.1.2 Data Fusion CSC Operational Description

Data Fusion can be divided into two segments, a user development segment, and a run time segment.
The user development segment involves:

- User off-line definition of Data Fusion Algorithms via the use of a Data Fusion Editor
- Integrate Test build of the Data Fusion Algorithm Table

The run time segment consists of:

- Loading the Data Fusion Algorithm Table during DDP initialization
- Applying data fusion on FDs received at the DDP by the Fusion processes
- Storing computation results in the CVT and making the data accessible by applications via Data Distribution.

1.2 Data Fusion CSC Specifications

1.2.1 Data Fusion CSC Ground Rules

1. Data Fusion inputs can be any valid FDs available in the data stream.
2. The maximum number of FDs used in a single calculation is 100.
3. The maximum number of FD samples in a single calculation is 200.
4. The user will specify Data Fusion FD data types using the defined engineering units data types (Refer to Appendix C).
5. Data Fused FDs shall be treated the same as any other FDs.
6. Fusion calculations/formulas may contain user-changeable coefficient.
7. Fusion FDs and Fusion algorithms cannot be created in a real-time environment. They must be edited/created prior to TCID build.
8. Data Fusion will not support user overriding Fusion output and inhibiting Fusion calculation for Redstone.
9. The user will associate a fusion algorithm with a Fusion name, or RSYS name.
10. There will be a single algorithm table per TCID.
11. Data Fusion CSC will make use of Data Distribution API to obtain and store fusion data from/to the CVT and / or queued services.
12. Data fusion data will be made available for applications access at the DDP, CCP, and HCI via Data Distribution.
13. Applications with the exception of Data Health manager and Data Fusion manager, will obtain fusion data via FD Services.
14. Data Fusion will be performed on a one-pass execution basis, so that processing is done with a cohesive set of data.
15. Data Fusion supports forward fusion only. Fusion output can not be used as Fusion input on the same processing cycle.
16. The user may alter user-changeable coefficient at real-time as needed to change the data fusion calculation but not the formula.

1.2.2 Data Fusion CSC Functional Requirements

The following Data Fusion Functional requirements are listed under two sections:

- Data Fusion Editor
- Data Fusion run time processing

The requirements for the Editor and run time processing will be used as specification for COTS evaluation. ControlShell was chosen as a viable COTS tool. ControlShell will be utilized as a run-time engine for Data Fusion processing. There are several requirements for the Data Fusion Editor that are not available through the component editor of ControlShell. The editor capability for Redstone will be done using a UNIX text editor. Details of the ControlShell editor requirements which are not available are italicized below.

1.2.2.1 Data Fusion Editor (DFE)

The italicized items in items 1, 2, and 3 above are not available in the ControlShell component editor. Items which are italicized but in bold are not available in ControlShell but can be accomplished with another editor. Italicized items will not be demonstrated for Redstone. The other items will be revisited in Thor and if the capability is required a solution which demonstrates the requirement will be delivered.

1. *DFE will provide an Graphical User Interface (GUI) allowing user to:*
    a. *select input FDs using a list of FDs in a user specified file*
    b. define fusion algorithms/formula using predefined operators list
    c. *select output FD*
    d. ***enter data validity requirement(s) of a fusion FD. User may specify formula for resultant health determination of the Fusion FD.***
    e. *query and search other data fusion FD and their compositions to help create new ones*
        - *Provide filtering options for search*
            i. *FD name*
            ii. *group of FDs*
            iii. *Fusion name*
            iv. *RSYS name*
            v. *Keyword*
    f. ***copy, cut, and paste of an existing Fusion algorithm to create a new algorithm***
    g. ***modify a fusion algorithm***
    h. ***save algorithm definition to an existing file***
    i. ***save algorithm definition to a new file (user defined)***
    j. ***exit without saving***
    k. *test the new algorithm (debug option)*
        - *Allow user to enter values of input FDs*
        - *Allow user to create value of user changeable variable*
        - *Output resulted FD value and status*
    l. *print FD and associated algorithm*
2. DFE will display error messages when:
    a. *A duplicate fusion output FD is entered. Upon which the user will be prompted with a "override" message.*
    b. *Inconsistent data types are entered*
3. DFE will provide On-Line Help

4. DFE will support prioritization of fusion algorithms
5. DFE will provide the following Arithmetic Operators:
    a. Addition (+)
    b. Subtraction (-)
    c. Multiplication (*)
    d. Division (/)
    e. Modulus
6. DFE will provide the following Relational Operators:
    a. Less Than (<)
    b. Greater Than (>)
    c. Less than or equal to (<=)
    d. Greater than or equal to (>=)
    e. Equivalent (=)
    f. Not equivalent (/=)
7. DFE will provide the following Logic Operators:
    a. AND
    b. OR
    c. XOR
    d. NOT
8. DFE will provide the following Bitwise Operators:
    a. AND
    b. OR
    c. XOR
    d. Left Shift
    e. Right Shift
    f. Ones complement
    g. Twos complement
9. DFE will provide the capability to support the “if then else” conditional operator either inside or outside of the equation/calculation
10. DFE will provide the capability to support the Power Function (a**b)
11. DFE will provide the capability to support the Exponential Function (e**x)
12. DFE will provide the capability to support the Square Root Function
13. *DFE will provide the capability to support bi-directional conversions:*
    a. *knots < = > miles per second*
    b. *degrees < = > radian*
    c. *BCD < = > Decimal*
    d. *hours < = > minutes*
    e. *Time of Day (TOD) < = > Greenwich Mean Time*
    f. *meters < = > feet*
    g. *miles < = > feet*
    h. *Centigrade < = > Fahrenheit*
    i. *pounds < = > grams*
    j. *pounds per sqare inch < = > pounds per sqaure foot*
    k. *gallon < = > litter*
14. DFE will provide the capability to support the following mathematical functions:
    a. Absolute
    b. *ln*
    c. Log10
15. DFE will provide the capability to support the following Trigonometric Operators:
    a. sin()
    b. cos()
    c. tan()
    d. cot()
    e. sec()
    f. csc()

    g. atan()
16. *DFE will support the Average function:*
    a. *for changed values.*
    b. *for values over time*
17. *DFE will support the Minimum/Maximum Function*
    a. *e.g., Given floating point value of 1.5 : min returns 1 and max returns 2*
    b. *e.g., Given a number of FDs : min returns the lowest and max returns the highest*
18. *DFE will support predefined constants:*
    a. *pi*
19. *DFE will support filtering operators:*
    a. *e.g., spike capture*
    b. *e.g., spike elimination*
    c. *e.g., digital filter*
20. *DFE will support data smoothing operator*
    a. *Sample rate reduction (e.g., 100 sample/sec to 1 sample/sec)*
21. *DFE will support data differentiation and integration operators:*
    a. *1^{st} derivative based on time*
    b. *integral based on time*
22. *DFE will support health operators:*
    a. *test if an FD is failed*
    b. *test if an FD is in warning condition*
    c. *test if 'x' of 'y' conditions are true (voting logic)*
23. *DFE will conform to operator procedures. The DFE will generate the Data Fusion Algorithm Table consisting of but not limited to the following information:*
    a. *input FD(s)*
    b. *output FD*
    c. *algorithm to generate out value*
    d. *algorithm to determine output health*
    e. *object name*
    f. *fusion name or RSYS name*

    ControlShell does not create a table and this capability will not be demonstrated in Redstone.

1.2.2.2 Data Fusion Manager (DFM) in the DDP

1. DFM will execute the operation defined in the Algorithm component.
2. DFM will make fused FDs available to be incorporated to the Data Distribution packets.
3. DFM will support prioritizing fusion algorithm execution.
4. DFM will provide the capability allowing user to control when fusion FDs are executed:
    a. *Executes whenever an input or a set of inputs is changed within a SSR (cycle).* This capability is an event driven algorithm. An algorithm can be designed to execute only once or twice during a run, but the algorithm must manage execution based on the trigger FD.
    b. Executes whenever a fusion process is scheduled.

    All algorithms will be executed in a cyclic fashion.

1.2.2.3 Data Fusion Manager Future Requirements

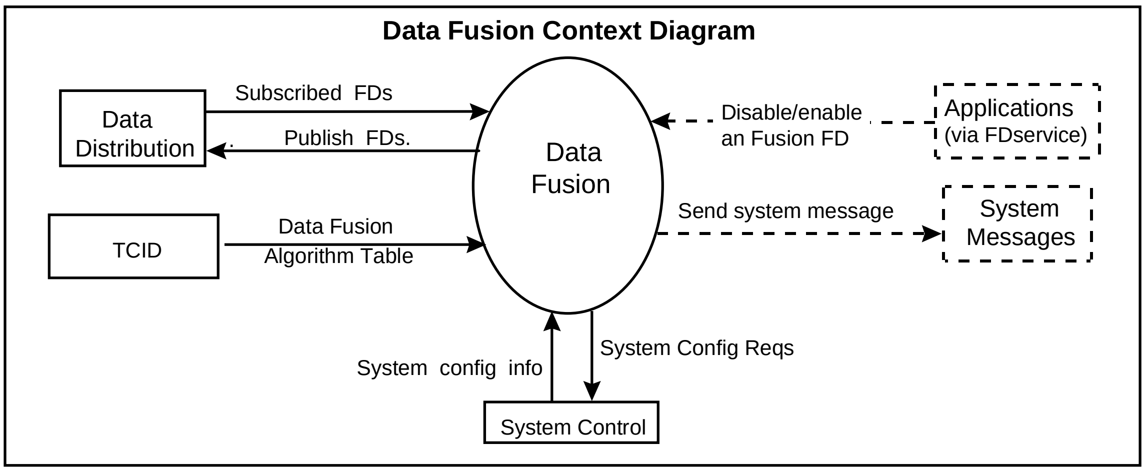
1. The Data Fusion Manager will capture run time errors and display them via an appropriate system messaging mechanism. (post Redstone).
2. The Data Fusion Manager will provide an application interface allowing application to:
    a. inhibit processing on any user selected Fusion FDs (post Redstone).
    b. activate processing on any user selected Fusion FDs (post Redstone).
    c. override a Fusion FD value. The changed value will be persistent. (post Redstone).

1.2.3 Data Fusion CSC Performance Requirements

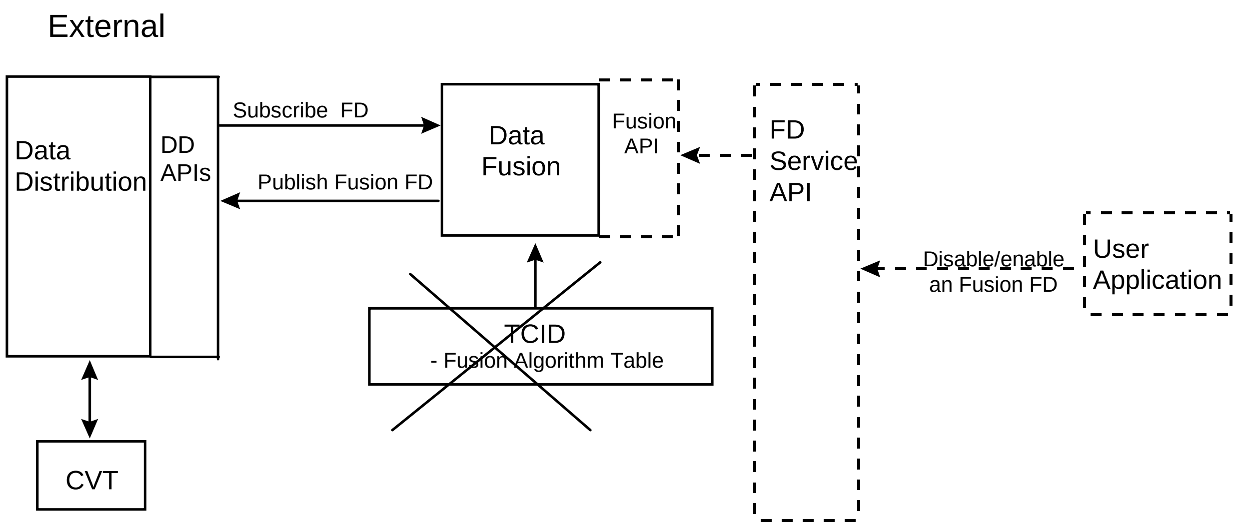
The following performance requirements are under RID review process, subject to change.

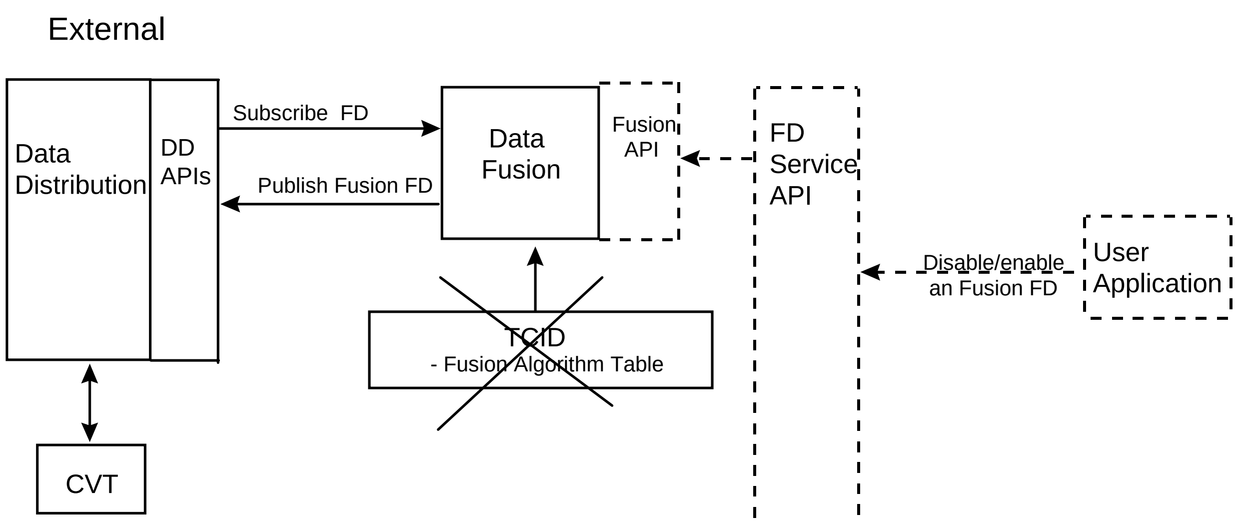
1. The Data Fusion will process up to the "system maximum data bandwidth" of FDs with one fusion calculation per change.
2. The Data Distribution shall support the "system maximum data bandwidth", plus 5,000 (20%) Data Fusion updates per second.

1.2.4 Data Fusion CSC Interfaces

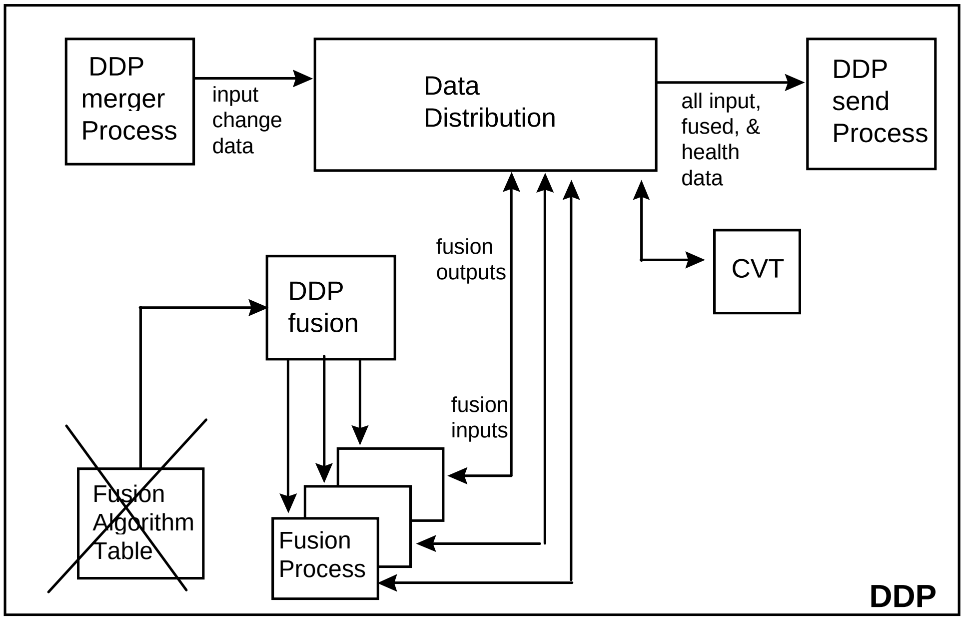


1.2.5 Data Fusion CSC Data Flow Diagram



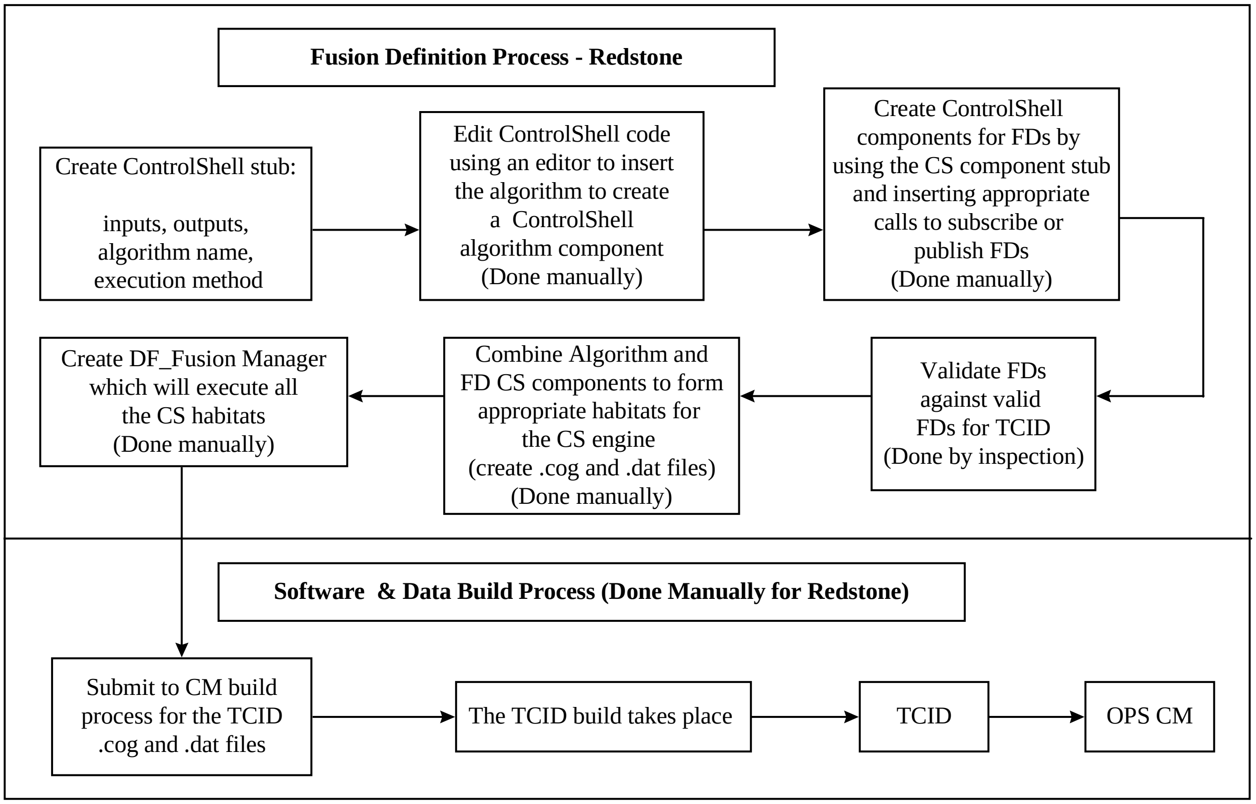


Internal


DDP merger Process
input change data
Data Distribution
all input, fused, & health data
DDP send Process
fusion outputs
CVT
DDP fusion
fusion inputs
Fusion Algorithm Table
Fusion Process
DDP

1.3 Data Fusion CSC Design Specification

The Data Fusion process begins with an off-line portion which builds the components. After the components have been created, they are submitted to the test build process to build a ControlShell application executable be included in a TCID. The fusion algorithms for the TCID are executed during a run. The fused FDs are then sent through a Data Fusion FD manager to be published as required.



Fusion Definition Process (Redstone)

The Data Fusion definition process begins with creating a ControlShell component stub. The ControlShell stub allows the user to create the inputs , outputs, algorithm name, and to identify the execution method. The algorithm must then be created using a UNIX editor and merged with the ControlShell stub. The resulting module is called a ControlShell algorithm component.

After the algorithm components have been created, ControlShell FD components for the algorithms are created. Creation of ControlShell FD components is done in a similar manner to creating an algorithm component. First create the stub, and then edit the stub to insert the appropriate calls to get the input FDs or publish the resulting fused FDs.

After the ControlShell Algorithm components and the ControlShell FD Components have been created, a validation of the FDs used must take place. This requires comparing the FDs input and output from the algorithm components to insure that the FDs are valid for the TCID. This process will be done by inspection for Redstone.

After the FDs have been validated against the TCID, the FD components and the algorithm components must be combined into units for execution. The combination of these components creates a sample habitat. Habitats are an execution unit, managed by the ControlShell engine, to execute the specified algorithms at a pre-set cycle rate. Each habitat can be run at a different cycle rate.

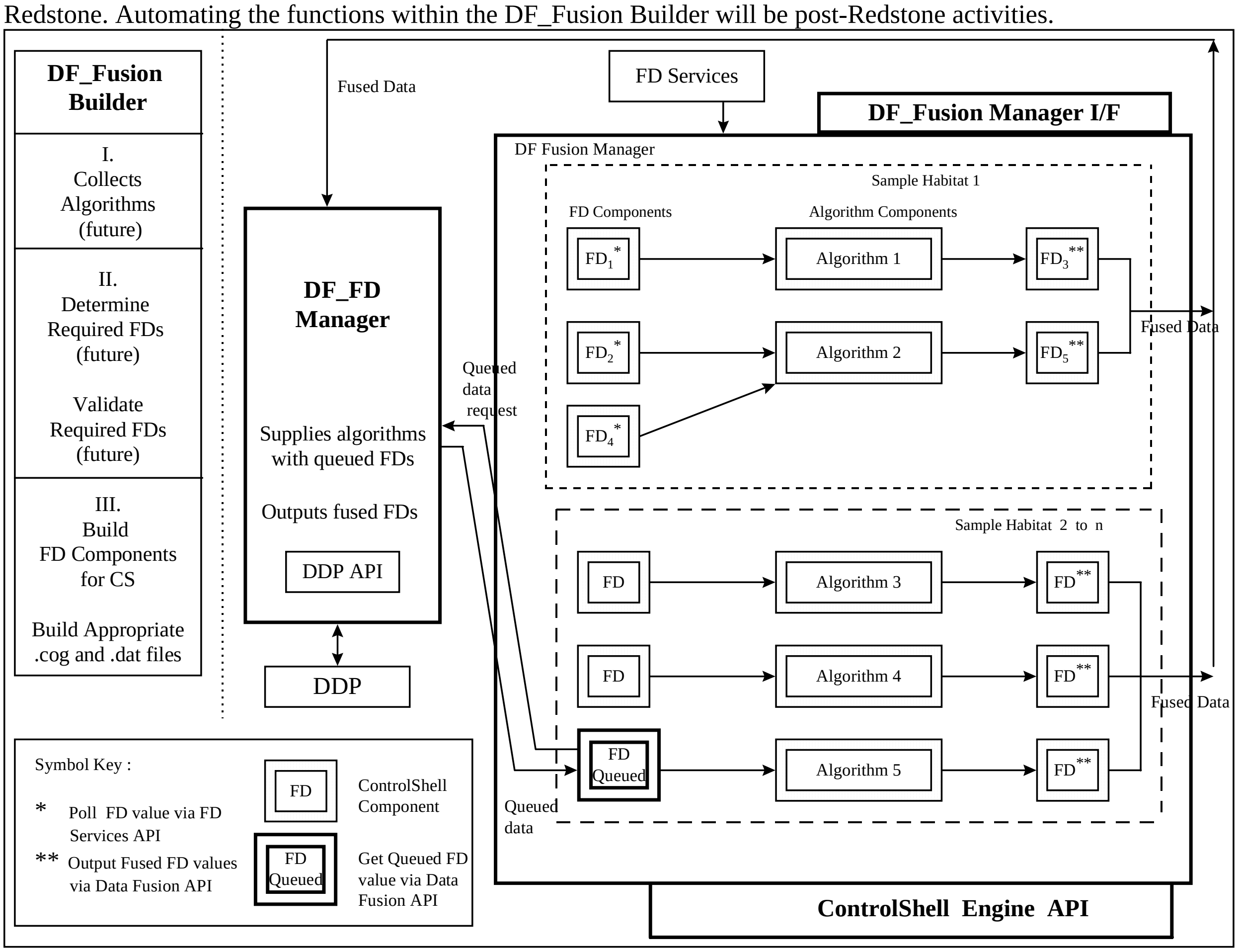
ControlShell requires that the habitats be merged into a file, so the ControlShell engine can execute each habitat at the desired cyclic rate. The ControlShell engine is built during compile time. The ControlShell engine uses the resulting .cog and .dat files to create the complete engine.

Finally, code must be added to the ControlShell engine to allow the Fusion process to start and stop during any given TCID. The combined code, the ControlShell engine and the start and stop mechanism forms the DF_Fusion Manager. ControlShell allows the cycle rates to be changed during a run, for any given habitat. It also allows for algorithms to be marked as executable or dormant. These capabilities will be available in post-Redstone releases. The DF_Fusion Manager must also have the code to alter execution rates of habitats and mark as executable or dormant. These capabilities will be in a post-Redstone release.

The Fusion definition process is defined in the data fusion general flow diagram as the DF_Fusion Builder (section 1.3.1). The components, habitats, and manager code, created by the DF_Fusion Builder are submitted to the TCID build process. For Redstone, this will be a manual process. Automation of some or all of the fusion definition process will be done in a post-Redstone release.

1.3.1 Data Fusion General Flow

The Data Fusion definition process, which will be done manually for Redstone, is depicted in the DF_Fusion Builder. Portions of the DF_Fusion Builder which require a validation process will be done by inspection for Redstone. Automating the functions within the DF_Fusion Builder will be post-Redstone activities.



The DF_FD Manger is custom built code which allows the fusion algorithms to access queued FDs and publish fused FDs. The DF_ FD Manager utilizes Data Distribution APIs to access or publish the required FDs.

The DF_Fusion Manager is a combination of custom built code and the ControlShell tool. It's main purpose is to perform the execution of the fusion algorithms during a specific run.

DF_Fusion Builder

The DF_Fusion Builder is the custom built code which completes the fusion definition process. Details of the definition process are listed in the previous section entitled Fusion Definition Process.

Once the algorithm components have been created, all the algorithms must be combined into appropriate habitats. This will be done manually for Redstone. Ways to automate the process of collecting all the algorithm components and combining them into habitats will be investigated in post-Redstone releases.

Determining the FDs required and validating the FDs against the valid FD list for a particular TCID is also discussed in the Fusion Definition Process section above. These processes will be done by inspection for Redstone. Automating the validation procedure will be investigated in post-Redstone releases.

The final portion of the DF_Fusion Builder is the building of the ControlShell components and inserting any additional code required to build the DF_Fusion Manager. Again, this process is described in detail in the above section, Fusion Definition Process. This portion of the DF_Fusion Builder will be done manually for Redstone. Automating this procedure will be investigated in post-Redstone releases.

Fusion Execution Run-Time Process

DF_FD Manager

The DF_FD Manager is custom built process which allows the fusion algorithms, being executed by the DF_Fusion Manager, to access queued FDs and to publish fused FDs. An algorithm which requires a queued FD will make a call to the DF_FD Manager API, ddf_get_queued_FD. The DF_FD Manager will determine the queue size, and collects all the FDs requested from different components then make a call to the Data Distribution API, ddp_get_queued_FD. The DF_ FD Manager will mange all requests for queued data and return the required FD values to the requesting FD component.

All fused FDs will be processed through a fused FD component which call the DF_FD Manager to publish the fused FD. The fused FD component will issue a ddf_publish_FD call. The DF_FD Manager will determine where the FD should be published and the call the Data Distribution API, ddp_publish_fusion_FD to distribute the fused FD as required.

DF_Fusion Manager

The DF_Fusion Manager is a ControlShell application which consists of a **sample habitat,** the **ControlShell engine,** and the **DF_Fusion Manager.**

The **sample habitat** consists of two distinct ControlShell components. The first component is the **FD component**. There is one FD component for each FD. There is also one FD component for each fused FD. The FD components make a call to an FD_Services API, asv_fds_get_FD, to read single value FDs. The FD component makes a call to the DF_FD Manager, using ddf_get_queued_FD to read any queued FD data. All fused FDs are output through a fused FD component. The fused FD component makes a call to the DF_FD Manager via ddf_publish_FD API. The second ControlShell Component within a **sample habitat** is the **algorithm component**. The algorithm component contains the code which represents the algorithm execution process. This is a ControlShell component which is built during the fusion definition process. The algorithm component does not call any APIs directly. Any FDs required for the algorithm execution are accessed through an FD component. The grouping of the algorithm components and the associated FD components become a sample habitat.

The **ControlShell engine** executes the sample habitats in a cyclic manner or an event. Each habitat may run on a different cycle time. A clock tick is issued each time a sample habitat executes. The clock tick can be a preset value or defined by the algorithm component developer. The clock tick definition is done by the DF_Fusion Builder during the fusion definition process.

The **DF_Fusion Manger** interface is the custom code which interfaces the ControlShell application with its outside world. It calls the ControlShell engine API and any additional directives which are required for the fusion execution. For Redstone, the start fusion and stop fusion directives would be the only additional code. ControlShell allows for changing the execution rate on a particular habitat as well as turning the execution on or off for a particular algorithm component within a habitat. The rate change and the specific algorithm component directives are changed through a handle to the habitats which are provided by the ControlShell tool. Utilization of these habitat change mechanisms may be demonstrated in a post-Redstone release.

1.3.2 Data Fusion External Interfaces

Data Fusion external interfaces are made through the FD components or by the DF_FD Manager. The FD components call FD Services to receive single value FDs. The call is to asv_fds_get_FD. Queued FD values and fused FDs are received or published by an internal interface calls within the Data Fusion CSC.

1.3.3 Message Formats

Message Number = 1
Message Group = DDF
Severity = Informational

DDF Manager is initialized

Help Information Content:
The Data Fusion Manager has successfully been initialized with the appropriate algorithms for this TCID.

Message Number = 2
Message Group = DDF
Severity = Critical

DDF Manager Unable to initialize

Help Information Content:
The Data Fusion Manager has could not be initialized with the appropriate algorithms for this TCID.

Message Number = 3
Message Group = DDF
Severity = Major

DDF Unable to complete algorithm

Help Information Content:
The Data Fusion Manager received an error from an algorithm indicating the algorithm terminated with an error condition

Message Number = 4
Message Group = DDF
Severity = Informational

DDF Manager is terminating

Help Information Content:
The Data Fusion Manager is terminating for this TCID.

1.3.4 Display Formats

None. (For Redstone)

1.3.5 Input Formats

None.

1.3.6 Recorded Data

None.

1.3.7 Printer Formats

None.

1.3.8 Interprocess Communications

None.

1.3.9 External Interface Calls (API calling formats)

asv_fds_get_FD
Gets a single FD value from the CVT.

ddf_get_queued_FD
Gets an FD which need to be queued for the executing algorithm from DF_FD Manager.

ddp_get_queued_FD
Returns queued data to DF_FD Manager.

ddf_publish_fusion_FD
Published fused FD value

1.3.10 Table Formats

None. (For Redstone)

1.3.3 Data Fusion Test Plan

1.3.3.1 Build Time Test Plan

A. Create several FD algorithm components utilizing actual algorithms available from various subsystems.
B. Put FD algorithms in CM repository
C. Submit for TCID build

1.3.3.1 Run-Time Test Plan

A. Run several cyclic FD algorithms. These algorithms will be sample algorithms from various subsystems which will exercise the functionality specified in the Data Fusion Editor portion of this document. Some algorithms may exercise more than one type of functionality. If the subsystem sample algorithms do not cover a particular specification, an test algorithm will be generated. It is expected that a subsystem algorithms will be able to cover all the available functionality within the Data Fusion Editor for Redstone. The subsystems from which we have sample algorithms are, SLWT, HMF, Fuel Cells, and Metro.
   1) At a specified rate
   2) At a different rate
   3) Verify the higher priority FD ran first

C. Generate a system message

Post Redstone Issues / Considerations

1. *Automated method to get fusion algorithms into the TCID build.*
2. *Store and view fusion algorithms via the system viewers.*
3. *Automate method to validate required FDs against the TCID.*
4. *Editor requirements for fusion algorithms.*

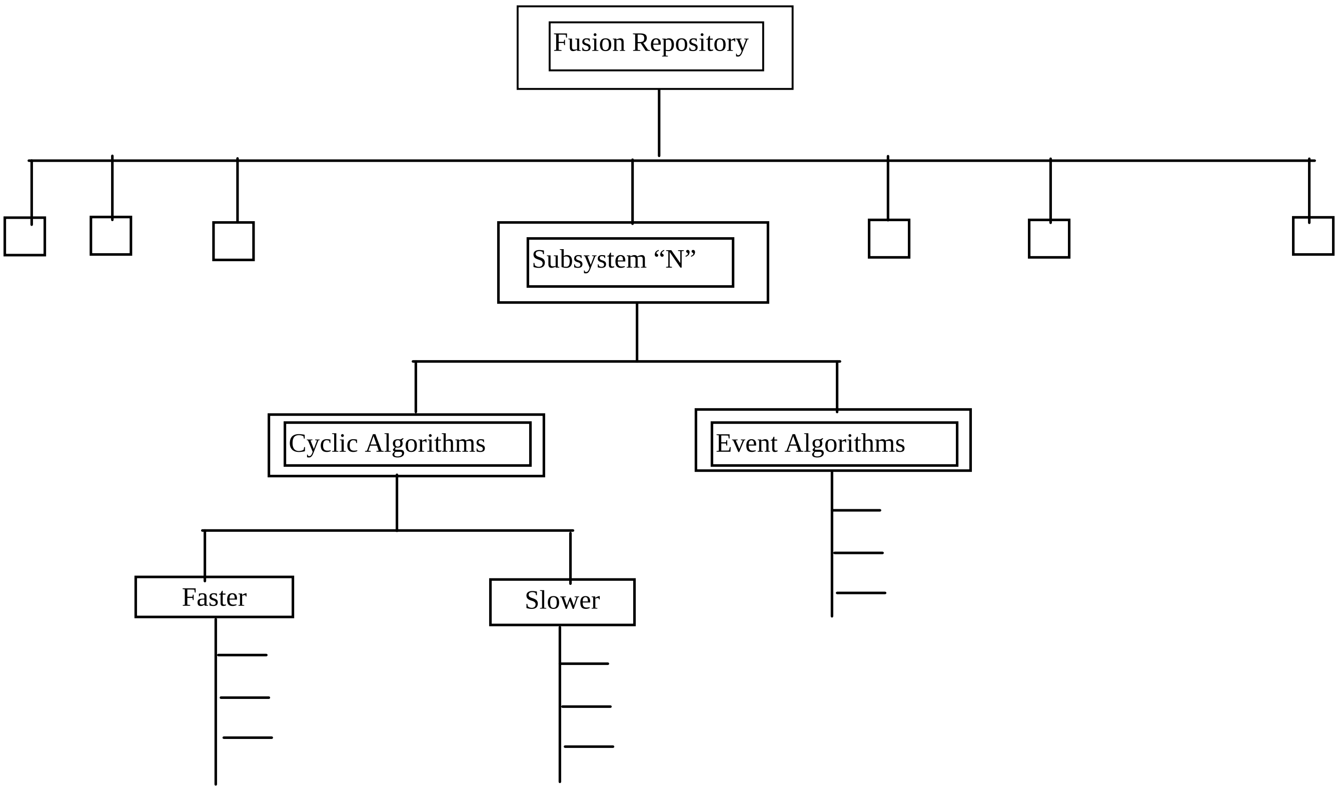
APPENDIX A

Statement of Work

- Define the list of logical and mathematical function required by the users for Data Fusion FDs.
- Define the list of logical and mathematical functions required by the users for Data Fusion Health.
- Define the Data Fusion FD types.
- Determine if a COTS tool can be utilized and provide the selected tool.
- Provide the initial Pre-build Data Fusion Editor.
- Provide the Data Fusion capability as part of the CLCS DDP Group.
- Provide the capability for Fused FD's to be utilized by the Data Distribution Manager for both the CCP and the HCI.
- Provide the capability to add fused FD's to the Data Bank.
- Provide the capability for Fused FD's in the Test Build process
- *Provide an initial System Viewer with the minimum capability to view fused FD's, including the Fused FD value, associated input FD values and the function being used to generate the Fused FDs.*
- Incorporate fused FDs into the record and retrieve capability with same capability as Gateways FD's and User Application Derived FD's.
- Provide the capability for the Data Fusion Function to be utilized in both Operational and Application environment (eg.. DDP & HCI/DDP/CCP/GW/MM Logical Subsystems).
- Provide performance data for system modeling
  Post-Redstone Activities
- *Provide capability to inhibit processing on individual Fusion FD's.*
- *Provide the capability for Fused FD's to be used utilized by the Constraint Manager Function.*

Appendix B

Possible Algorithm Repository Structure



The Fusion Algorithm Repository will reside in *TBD*. User who create algorithms will put the approved algorithms in the Fusion Algorithm Repository. Each subsystem will be able to manage how their fusion algorithms are executed. The algorithms should be divided into two categories. Cyclic algorithms execute at a particular rate. All algorithms listed in a particular group will execute at the same rate. The rate may be changed by issuing a rate change from an application. (details TBD). If a subsystem requires several algorithms to be executed at a particular rate and another group of algorithms executed at a different rate, they will put them in two sub-directories labeled appropriately. For illustration purposes, they are labeled *Faster* and *Slower* in the diagram. A subsystem may change the rate of execution or either group of algorithms. The rate may be increased or decreased as required. The subsystem may also execute event driven algorithms. It is assumed that the event driven algorithms will only execute once or twice a run. The event driven algorithms which are required to run multiple times should be placed in the cyclic directory.

The Fusion API will execute a UNIX script to traverse the Fusion Algorithm Repository collecting the appropriate algorithms for the TCID.